대 한 민 국 특 허 청
KOREAN INTELLECTUAD PROPERTY OFFICE

별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto is a true copy from the records of the Korean Intellectual Property Office.

출 원 번 호 : 10-2003-0016208
Application Number

출 원 년 월 일 : 2003년 03월 14일
Date of Application MAR 14, 2003

출 원 인 : 주식회사 안철수연구소 외 1명
Applicant(s) Ahnlab, Inc., et al.

2003 년 04 월 29 일

특 허 청
COMMISSIONER

【서지사항】

【서류명】 특허출원서
【권리구분】 특허
【수신처】 특허청장
【참조번호】 0005
【제출일자】 2003.03.14
【국제특허분류】 H04L
【발명의 명칭】 코드 삽입 기법을 이용한 악성 스크립트 감지 방법
【발명의 영문명칭】 METHOD TO DETECT MALICIOUS SCRIPTS USING CODE INSERTION TECHNIQUE

【출원인】
【명칭】 주식회사 안철수연구소
【출원인코드】 3-1999-902882-2

【출원인】
【성명】 홍만표
【출원인코드】 4-1999-008549-0

【대리인】
【성명】 박창남
【대리인코드】 9-2001-000437-2
【포괄위임등록번호】 2003-015642-9
【포괄위임등록번호】 2003-016061-0

【대리인】
【성명】 진천웅
【대리인코드】 9-1998-000533-6
【포괄위임등록번호】 2003-015641-1
【포괄위임등록번호】 2003-016062-7

【발명자】
【성명의 국문표기】 이성욱
【성명의 영문표기】 LEE,Sung Wook
【주민등록번호】 690408-1018827
【우편번호】 440-320
【주소】 경기도 수원시 장안구 율전동 샘내마을삼호아파트 211-906
【국적】 KR

【발명자】

【성명】 홍만표

【출원인코드】 4-1999-008549-0

【발명자】

【성명의 국문표기】 조시행

【성명의 영문표기】 CHO,Si Haeng

【주민등록번호】 620219-1029514

【우편번호】 140-040

【주소】 서울특별시 용산구 산천동 192 리버힐삼성아파트 109-1102

【국적】 KR

【공개형태】 간행물 발행

【공개일자】 2002.12.15

【심사청구】 청구

【취지】 특허법 제42조의 규정에 의한 출원, 특허법 제60조의 규정에 의한 출원심사 를 청구합니다. 대리인
박창남 (인) 대리인
진천웅 (인)

【수수료】

【기본출원료】 20 면 29,000 원

【가산출원료】 6 면 6,000 원

【우선권주장료】 0 건 0 원

【심사청구료】 2 항 173,000 원

【합계】 208,000 원

【감면사유】 중소기업

【감면후 수수료】 104,000 원

【첨부서류】 1. 요약서·명세서(도면)_1통 2.중소기업기본법시행령 제2조에의한 중소기업에 해당함을 증명하는 서류_1통 3.공지예외적용대상(신규성상실의예외, 출원시의특례)규정을 적용받기 위한 증명서류_1통

【요약서】

【요약】

본 발명은 코드 삽입 기법을 이용한 악성 스크립트 감지 방법에 관한 것이다. 이러한 본 발명은 매칭 룰(Matching Rules)과 관계 룰(Relation Rules)을 포함한 룰(Rules) 기반의 메소드 호출 시퀀스 탐지를 채용하여 호출 시퀀스에 속한 각각의 문장에 관련된 값들을 검사하되, 원본 스크립트의 메소드 호출 문장 전후에 자체 진단 루틴(악성 행위 감지 루틴) 호출 문장을 삽입하는 단계; 및 상기 원본 스크립트에 삽입된 자체 진단 루틴을 통해 실행시 악성 코드를 감지하는 단계를 포함한 것을 특징으로 한다.

본 발명에 따르면, 감지 루틴이 스크립트 실행 중에 동작함으로써 동적으로 결정되는 리턴값과 파라미터까지 검사할 수 있어서 감지 정확도를 향상시킬 수 있다. 또한, 외부로부터 유입된 스크립트에만 코드가 삽입됨으로써 불필요한 오버헤드를 발생시키지 않으며, 일단 변형된 코드는 별도의 안티바이러스가 설치되지 않은 시스템에서도 자체 감지를 수행함으로써 전파 억제 효과를 얻을 수가 있다.

【대표도】

도 4

【색인어】

컴퓨터 바이러스, 코드, 룰, 어플리케이션, 감지, 스크립트, 메소드

【명세서】

【발명의 명칭】

코드 삽입 기법을 이용한 악성 스크립트 감지 방법{METHOD TO DETECT MALICIOUS SCRIPTS USING CODE INSERTION TECHNIQUE}

【도면의 간단한 설명】

도 1 은 메일을 통하여 자기 복제를 수행하는 비주얼 베이직 스크립트의 실예,

도 2 는 메일을 통한 자기 복제 행위 정의의 실예,

도 3 은 어플리케이션 변환 시스템에 대한 개념도,

도 4 는 본 발명을 설명하기 위한 개념도,

도 5 는 본 발명에 따라 악성 스크립트를 감지하는 과정을 나타내는 흐름도,

도 6 은 본 발명에 따른 룰 기술 문법에 대한 실예,

도 7 은 비주얼 베이직에서 전자우편을 통해 추출된 자기 복제 패턴을 나타낸 개념도,

도 8 은 전자우편을 통한 자기 복제 패턴 감지를 위한 관계 룰 정의에 대한 실예,

도 9 는 코드 삽입 후 메소드 호출 코드의 실예,

도 10 은 악성 행위 감지 루틴에 의해 스크립트의 실행 중에 이루어지는 룰 인스턴스의 생성의 실예이다.

*도면의 주요부분에 대한 부호의 설명

2 : 원본 스크립트 4 : 진단 룰

6 : 변환된 스크립트 10 : 자체 진단 루틴 생성기

20 : 스크립트 변환기

【발명의 상세한 설명】

【발명의 목적】

【발명이 속하는 기술분야 및 그 분야의 종래기술】

<15> 본 발명은 악성 스크립트 감지 방법에 관한 것으로서, 특히 코드 삽입 기법을 이용하여 지속적인 행위 감시를 통해 악성 코드를 감지할 수 있는 기술에 관한 것이다.

<16> 악성 코드(malicious code)는 비정상적인 동작 또는 시스템 위해(harm) 행위를 목적으로 작성된 코드를 말하며, 컴퓨터 바이러스(computer virus), 웜(worm), 그리고 트로이 목마(trojan)를 포함하는 개념이다. 악성 스크립트는 스크립트 언어로 작성된 악성 프로그램들을 말하는데, 현재까지 발견된 것들은 비주얼 베이직 스크립트(Visual Basic Script), mIRC 스크립트, 자바 스크립트가 수적으로 가장 많으며, 그 외에 PHP 스크립트, 코렐 드로우 스크립트 등으로 작성된 것들이 일부 존재한다.

<17> 이러한 악성 스크립트의 감지에는 이진 형태의 악성 코드와 마찬가지로 시그너쳐(signature) 기반의 스캐닝(scanning)을 통한 방법이 보편적으로 사용되고 있다. 그러나, 이러한 기법은 사전에 면밀한 분석을 통해 시그너쳐를 추출한 악성 코드만을 감지할 수 있으므로, 알려지지 않은 새로운 악성 스크립트의 감지에는 휴리스틱(heuristic) 스캐닝, 정적 분석, 행위 감시 기법 등이 사용된다.

<18> 본 발명의 이해를 돕기 위해 종래의 알려지지 않은 악성 스크립트 감지 기법을 살펴보기로 한다.

<19> 첫째, 휴리스틱 스캐닝이란 악성 행위를 위해 자주 사용되는 메소드(method) 또는 내장 함수(intrinsic function) 호출들을 데이터베이스화 하여두고 대상 스크립트를 스캔하여 일정 수 이상의 위험한 호출이 나타나면 이것을 악성 스크립트로 간주하는 방식이다. 이 방식은 속도가 비교적 빠르고 높은 감지율을 보이긴 하지만 악성이 아닌 선의의 (legitimate) 스크립트를 악성으로 감지하는 긍정 오류(false positive)가 상당히 높다는 큰 단점을 가지고 있다.

<20> 둘째, 정적 분석 기법은 이런 단점을 극복하기 위해 각각의 위험한 메소드 호출이 아니라 악성 행위를 구성하는 메소드 시퀀스들을 정의함으로써 악성 행위를 정확하게 감지하려는 의도에서 제안되었다. 도 1 은 이러한 정적 분석 기법을 설명하기 위한 메일을 통하여 자기 복제를 수행하는 비주얼 베이직 스크립트의 실예이다. 다수의 메소드 호출이 하나의 악성 행위를 구성하기 위해서는 반드시 그것들의 파라미터와 리턴값 사이에 특별한 관계가 존재해야 함을 확인할 수 있다. 예컨대, 4행의 Copy 메소드는 현재 실행 중인 스크립트를 'LOVE-LETTER-FOR-YOU.TXT.VBS' 라는 이름으로 복사하고, 7행의 Attachments.Add 메소드는 그 파일을 새로 만들어진 메일 객체에 첨부함으로써 메일을 통한 자기 복제를 달성한다. 그러나, 메소드 호출의 존재유무만을 검사하는 방식을 사용하게 되면, A라는 이름으로 스크립트 파일을 생성하고 B라는 이름의 파일을 첨부하는 관계없는 메소드 호출이 존재하여도 이를 악성 코드로 간주하므로 높은 긍정 오류를 보이게 된다. 이 기법은 메소드 호출의 존재 뿐만 아니라, 상술한 파일명, fso, c, out,

male 등 모든 관계 있는 값들이 일치하는가를 검사함으로써, 다른 방식에서 나타나는 높은 오류율을 극복하려고 시도하였다.

<21> 실제에 있어서, 이러한 악성 행위는 단순히 일련의 메소드 시퀀스로만 정의할 수 없으며, 다양한 메소드 또는 메소드 시퀀스들의 조합으로 이루어진다. 따라서, 이 기법에서는 악성 행위가 단위 행위들의 조합으로 이루어지며, 각각의 단위 행위는 더욱 작은 단위 행위 또는 하나 이상의 메소드 호출들로 이루어진다고 모델링하고, 각 단위 행위와 메소드 호출 문장을 하나의 룰(rule)로 표현하였다. 예컨대, 도 1 에 나타난 메소드들만을 고려하여 악성 행위의 패턴을 정의하면 도 2 와 같은 형태로 표현할 수 있다. 즉, 도 2 는 메일을 통한 자기 복제 행위 정의의 실예이다. 도 2 와 같이, 룰은 매칭 룰(matching rule)과 관계 룰(relation rule)의 두 가지 종류가 있으며, 각각의 이름 첫 자가 M, R 인 것으로 구분된다. 매칭 룰의 경우에는 우측에 기술한 것과 동일한 패턴을 가진 문장이 존재하면 조건이 만족되고, 관계 룰의 경우에는 우측의 조건식이 참(true)이면 만족된다.

<22> 이러한 정적 분석을 통해 악성 행위 여부를 판단하는 것은 악성 행위에 사용될 수 있는 메소드 호출의 출현 빈도만을 고려하는 단순한 감지 기법에 비해, 극히

낮은 긍정 오류를 보장받을 수 있다는 장점을 가진다. 그러나, 궁극적으로 실행 전의 소스 코드 분석만으로는 실행시 해당 파라미터 또는 리턴값이 어떤 값을 가질지 예측할 수 없는 경우가 빈번하게 발생하므로, 실제로 악성 코드임에도 불구하고 이를 감지하지 못하는 부정 오류(false negative)가 높아질 가능성도 가지게 된다. 즉, 악성 행위의 메소드 호출 시퀀스를 통해 정확한 감지를 시도하나 실행시에만 결정할 수 있는 값이 하나라도 개입되면 다른 조건을 만족하여도 이를 악성행위로 간주할 수 없으므로 높은 부정 오류를 수반하게 된다.

<23> 셋째, 행위 감시 기법은 프로그램 수행에 필요한 시스템 호출들을 가로채어 감시하다가 악성행위로 판단되는 시스템 콜의 시퀀스가 나타나면 해당 프로그램을 악성 코드로 간주하는 감지 방식이다. 이 방식은 실행 시간 중에 감지를 행하므로 해당 코드의 정확한 수행 경로 추적이 가능하고 관련된 동적 데이터를 이용할 수 있다는 장점이 있다. 그러나, 이 기법은 모든 클라이언트에 행위 감시기가 설치되어야 하며, 실행 중인 모든 프로그램에 대한 감시로 인해 발생하는 오버헤드가 크다는 단점을 가지고 있다. 즉, 전자우편과 같이 특정 도메인 내에 진입하는 모든 자료가 하나의 서버를 거치는 서비스에서는 안티바이러스 시스템의 물리적인 설치 위치에 따라 서버측 대응 기법과 클라이언트 상에서의 대응 기법으로 바이러스 대응 기법을 분류할 수 있다. 서버 수준의 안티바이러스는 특정 도메인 내에 진입하는 악성코드를 진입점에서 차단하므로 모든 클라이언트를 완벽하게 통제하기 어려운 실제 상황에서 전자우편 서버 등에 유용하게 사용된다. 이 때, 서버에서의 악성 코드 감지를 위한 별도의 기법이 존재하지는 않으며, 이미 알려진 대응 기법을 서

버에서의 동작에 적절하도록 약간의 수정을 가하여 이용하는 것이 보편적이다. 그러나, 각각의 클라이언트에 설치된 감시 도구를 기반으로 동작하는 행위 감시 기법은 서버에 사용할 수 없으며, 에뮬레이션을 통한 가상 환경에서의 실행은 가능하나 서버에 많은 부담을 주므로 현실적으로 사용이 어렵다는 문제점이 있다.

<24> 이와 같은 종래 기법들의 문제점으로 인해, 현재 전자우편 서버 등에 탑재되는 서버용 안티바이러스는 시그너쳐 기반의 스캐닝을 기반으로 동작하며, 이에 필터링 또는 파일명 변환 등의 기능을 추가하여 알려지지 않은 새로운 악성 스크립트의 확산을 늦추는 소극적인 형태를 취하는 데에 그치고 있다.

【발명이 이루고자 하는 기술적 과제】

<25> 이에 본 발명은 상기와 같은 문제점을 해결하기 위하여 안출된 것으로서, 원본의 스크립트 코드에 자체 진단을 수행할 수 있는 스크립트 코드들을 삽입하고 해당 스크립트가 실행될 때 외부의 도움 없이 자신의 악성 여부를 판단할 수 있는 코드 삽입 기법을 이용한 악성 스크립트 감지 방법을 제공하는데 그 목적이 있다.

<26> 상기와 같은 목적을 달성하기 위하여 본 발명에 따른 코드 삽입 기법을 이용한 악성 스크립트를 감지하는 방법은, 매칭 룰(Matching Rules)과 관계 룰(Relation Rules)을 포함한 룰(Rules) 기반의 메소드 호출 시퀀스 탐지를 채용하여 호출 시퀀스에 속한 각각의 문장에 관련된 값들을 검사하되, 원본 스크립트의 메소드 호출 문장 전후에 자체 진단 루틴(악성 행위 감지 루틴) 호출 문장을 삽입

하는 단계; 및 상기 원본 스크립트에 삽입된 자체 진단 루틴을 통해 실행시 악성 코드를 감지하는 단계를 포함한 것을 특징으로 한다.

<27> 특히, 상기 자체 진단 루틴 호출 문장은, 상기 메소드 호출 문장이 상기 매칭 룰에 기술된 내용과 일치시 메소드 호출 문장 전후에 삽입되는 파라미터와 리턴값 저장, 및 진단 엔진을 호출하는 문장들로 구성되며, 상기 자체 진단 루틴은, 상기 매칭 룰과 일치하는 형태의 메소드 호출시 수행되어 해당 매칭 룰에 관련된 관계 룰을 실행하여 메소드 호출 시퀀스의 악성 행위 구성 여부를 탐지하는 룰 기반의 진단 엔진과, 상기 진단 엔진이 사용할 수 있는 버퍼에 상기 매칭 룰을 만족하는 메소드 호출 문장의 파라미터와 리턴값을 저장하는 메소드들을 포함하는 것이 바람직하다.

【발명의 구성 및 작용】

<28> 이하, 첨부된 도면을 참조하여 본 발명을 상세히 설명하기로 한다.

<29> 먼저, 본 발명을 구현하는데 변형하여 적용되는 어플리케이션 변환 기법에 대해 살펴보기로 한다. 어플리케이션 변환 기법은 코드 안전(code safety)을 위해 제안된 것으로, 실행시 안전을 확신할 수 없는 코드가 주어지면 사전에 정의된 정책(policy)을 강행할 수 있는 형태로 해당 코드를 변환하는 기법이다. 따라서, 변환이 완료된 코드를 실행하면 각각의 API가 호출될 때마다 해당 API 호출로 인해 접근하게 되는 시스템 자원이 허가되어 있는가를 검사한 후 원래의 작업을 수행하게 된다.

<30> 도 3 은 이러한 어플리케이션 변환 시스템에 대한 개념도이다. 도면을 참조하면, 전체 시스템은 크게 정책 생성기(policy generator)와 어플리케이션 변환기(application

transformer)로 구성된다. 정책 생성기는 최초 시스템 설치시 또는 보안 정책 변경시에 동작한다. 그리고, 이 때 입력으로는 시스템 리소스에 대한 추상적인 기술과 리소스 조작에 대한 제한 사항을 담고 있는 보안 정책(safety policy), 그리고 해당 플랫폼의 API 라이브러리와 이들의 리소스 사용 내역에 대한 정보가 주어진다. 따라서, 이러한 입력을 바탕으로 정책 강행에 필요한 코드를 삽입한 플랫폼 라이브러리(policy-enforcing platform library)와 실제적인 코드 수정 지침이 기술된 정책 기술 파일(policy description file)이 생성되면 어플리케이션 변환을 위한 준비 작업이 완료된다. 대상 코드가 주어지면 어플리케이션 변환기는 정책 기술 파일을 참조하여 해당 코드의 특정 API 호출을 변형된 플랫폼 라이브러리에 대한 호출로 교체함으로써 실행시에 사전 정의된 정책이 적용되도록 한다. 이러한 어플리케이션 변환 기법은 소스 또는 P-코드와 같이 소스 프로그램에 준하는 형태를 가진 이동 코드(mobile code)에 대한 접근 제어 강행에는 유용하게 사용될 수 있다. 그러나, 각각의 함수 호출간의 관계를 고려하는 것이 아니고 단지 특정 API의 실행 허가 여부만을 결정하므로 악성 행위의 패턴을 감지할 수는 없다.

<31> 도 4 는 본 발명을 설명하기 위한 개념도로서, 감지 방식에 있어서는 정적 분석 기법과 같은 룰(rule) 기반의 메소드 호출 시퀀스 탐지를 채용하되, 어플리케이션 변환 기법을 이용하여 탐지 루틴을 스크립트 소스에 삽입함으로써 실행시에 악성 코드를 감지하는 기법에 대한 것이다. 도면을 참조하여 설명하자면, 자체 진단 루틴 생성기(10:Self-Detection Routine Generator)는 매칭 룰(Matching Rules)과 관계 룰(Relation Rules)을 포함한 진단 룰(4:Detection Rules)을 기반으로 하여 악성 행위를 감지할 수

있는 자체 진단 루틴(악성 행위 감지 루틴)을 생성한다. 스크립트 변환기(20:Script Transformer)는 메소드 호출(Method call) 문장을 포함한 원본 스크립트(2)를 진단 룰(4)을 기반으로 하는 메소드 호출 시퀀스와 자체 진단 루틴 생성기(10)에서 생성된 자체 진단 루틴을 통해서 실행 중에 지속적으로 자체 진단을 수행할 수 있는 스크립트(6)로 변환시킨다. 즉, 외부로부터 유입되거나 악성 여부가 의심되는 스크립트를 실행 전 임의의 시점에 실행 중 지속적으로 자체 진단을 수행할 수 있는 형태로 변환시킨다. 이때, 스크립트 변환기(20)는 원본 스크립트(2)에 기술되어 있는 문장 자체를 변경하지는 아니하며 추가적인 코드의 삽입만을 수행한다.

<32> 한편, 도 5 를 참조하여, 도 4 와 같은 개념도를 바탕으로 매칭 룰(Matching Rules)과 관계 룰(Relation Rules)을 포함한 룰(Rules) 기반의 메소드 호출 시퀀스 탐지를 채용하여 호출 시퀀스에 속한 각각의 문장에 관련된 값들까지 검사하여 악성 스크립트를 감지하는 과정을 설명하기로 한다. 먼저 원본 스크립트의 메소드 호출 문장 전후에 자체 진단 루틴 호출 문장을 삽입한다(S510). 따라서, 원본 스크립트에 삽입된 자체 진단 루틴을 통해 실행시에는 악성 코드를 감지할 수가 있다(S520).

<33> 이번에는 삽입되는 코드에 대해 살펴보기로 한다. 첫째로, 원본 스크립트(2)의 메소드 호출(method call) 문장 전후에 파라미터와 리턴값을 취하고 자체 진단 루틴을 호출하는 문장들이다. 즉, 도 4 의 변환된 스크립트(6) 중에서 'put parameters to buffer', 'put return value to buffer', 및 'run Self-Detection Routine' 로 기술되었으며 실제로는 도 9 와 같은 형태를 가지게 된다. 이것은 모든 메소드 호출 문장에 삽입되는 것이 아니며, 매칭 룰에 기술된 것과 정확히 일치하는 형태를 보이는 메소드 호출 문장 전후에만 삽입된다. 이때, 스크립트 변환기(20)는 매칭 룰을 적절히 분석하여 필요

한 값들만을 취하고 자체 진단 루틴을 호출하는 코드를 삽입한다. 이는 메소드에 따라 취해야 할 파라미터의 수가 다르고, 리턴값이 없는 메소드 호출도 존재하기 때문이다.

<34> 둘째로, 자체 진단을 수행하는 루틴(Self-Detection Routine)이다. 이것은 스크립트의 내용과 무관하며, 오직 악성 행위를 정의한 룰에 따라서만 달라진다. 즉, 이 루틴은 악성 행위 정의가 갱신되지 않는 한 변경되지 않으므로, 자체 진단 루틴 생성기(10)에 의해 사전에 생성되며, 룰 기반의 진단 엔진(detection engine)과 버퍼 조작 메소드들(buffer handling method)로 구성된다. 진단 엔진은 매칭 룰과 일치하는 형태의 메소드 호출시에만 수행되므로, 해당 매칭 룰에 관련된 모든 관계 룰들을 실행하여 현재까지의 메소드 호출 시퀀스가 악성 행위를 구성하였는가를 검사하는 역할을 담당한다. 버퍼 조작 메소드는 매칭 룰을 만족하는 메소드 호출 문장의 파라미터와 리턴값을 진단 엔진이 사용할 수 있는 버퍼에 저장하는 메소드를 의미한다.

<35> 이러한 감지 기법은 특정 스크립트 언어에 국한되지 않으며 단지 사용하는 룰 집합만을 다르게 정의함으로써 다수의 스크립트 언어에 동일한 알고리즘을 적용할 수 있다. 특히, 마이크로소프트 윈도우즈에서는 비주얼 베이직 스크립트, 자바스크립트와 같은 다수의 언어가 동일한 윈도우즈 스크립팅 호스트를 통해 실행되며 동일한 런-타임(run-time) 객체 및 환경을 사용한다. 따라서, 대부분의 경우 각각의 스크립트 문법에 맞도록 매칭 룰만을 수정하는 것만으로 서로 다른 언어를 위한 룰 집합을 정의할 수가 있게 된다.

<36> 이제, 본 발명의 이해를 돕기 위해서 상술한 코드 삽입 기법을 구체화 할 때 고려해야 할 점들과 구현 내용을 설명하기로 한다. 먼저, 악성 행위의 정의에 사용되는 룰은 종래의 코드 정적 분석 기법의 것과 유사하나 좀 더 일관성이 있으면서 단순한 형태로

변경되는 것이 바람직하다. 도 6 은 본 발명에 따른 룰 기술 문법에 대한 실예를 나타낸다. 도 6 을 참조하면, 하나의 룰 기술 파일은 다수의 룰 정의로 구성되며, 각각의 룰은 룰 ID(rule_identifier)와 룰 바디(rule_body)로 구성된다. 룰 바디의 형식은 매칭 룰과 관계 룰에 따라 달라지는데, 매칭 룰의 경우에는 룰 변수(variable_string)를 포함하는 것 외에는 일반적인 스크립트 문장의 형태를 가지게 된다. 또한, 관계 룰의 경우에는 조건절(condition_phrase)과 동작절(action_phrase)로 구성되어, 조건절의 조건이 만족될 경우 동작절의 내용을 수행하게 된다. 조건절은 하나 이상의 조건식으로 구성되며, 각각의 조건식은 특정 룰이 이미 만족되었는지 또는 두 룰의 특정 변수값들이 같거나 다른 한편에 포함되는지를 검사하도록 기술된다.

<37> 그런데, 종래의 룰 기술을 복잡하게 만들었던 가장 큰 요인은 AND, OR와 같은 논리 연산자(logical operator)를 지원하지 않는 점이었다. 따라서, A, B, C가 각각 하나의 조건식(condition_expr)이라 하였을 때, '(A AND B) OR C' 와 같은 조건은 다음과 같이 기술되어야 했다.

<38> R1 : cond A

<39> R2 : precond A

<40> cond B

<41> action $global = true

<42> R3 : cond C

<43> action $global = true

<44> 즉, AND로 연결된 조건은 다수의 룰로 분리하여 사전 만족 조건절(pre-condition phrase)에 기술하고, OR로 처리해야 할 부분은 각각의 조건식이 만족될 경우 false로 초기화된 전역변수를 true로 변경하는 우회적인 방법으로 기술되었다. 이것은 룰 기술과 추후 해독을 어렵게 하는 요소이므로, 조건식에 논리 연산자를 직접 사용할 수 있도록 하고 전역 변수의 사용을 금지함으로써, 전체적으로 룰 기술이 단순하고 악성 행위의 논리와 일치하도록 수정하였다.

<45> 한편, 실제적인 룰 정의는 종래의 악성 코드들에 대한 경험적인 분석을 통해 얻어질 수 있다. 이론적으로 증명된 바와 같이 가능한 모든 악성 행위를 정확하게 감지할 수 있는 휴리스틱 룰의 집합은 존재하지 않으며, 새로운 악성 코드 또는 행위 패턴이 등장할 때마다 지속적인 갱신이 이루어져야 한다. 악성 코드가 일반적인 프로그램과 확연하게 구분될 수 있는 특성이 자기 복제(self-duplication)의 수행이라는 점에 주목하여, 현재까지 알려진 악성 스크립트의 자기 복제 행위를 표 1 과 같이 정리할 수 있다.

<46> 【표 1】

구 분	내 용
로컬 시스템 내의 자기 복제	대상 시스템에 자신의 복사본을 만든다. 엄밀히 구분하면, 존재하지 않는 새로운 복사본을 생성하는 경우와, 대상 시스템에 이미 존재하던 스크립트 파일의 내용을 자신과 같은 내용으로 바꾸어 넣는 행위로 나눌 수 있다.
전자우편을 통한 자기 복제	주소록에 기재된 계정으로 자신을 첨부한 전자우편을 전송한다.
IRC를 통한 자기 복제	mIRC와 같은 IRC 클라이언트의 초기화 스크립트를 변경하여 대화 상대 접속시, 자신을 전송하도록 한다.
공유 폴더를 통한 자기 복제	네트워크 공유 폴더를 검색하여 자신을 복사한다.

<47> 표 1 과 같은 각각의 악성 행위는 하나 이상의 단위 행위 또는 메소드 호출 패턴으로 정의되고, 각각의 단위 행위는 다시 하나 이상의 단위 행위 또는 메소드 호출 패턴으로 정의되므로, 특정 악성 행위는 하나의 룰이 하나의 노드로 나타나는 트리 형태로 표

현될 수 있다. 따라서, 알려진 많은 악성 스크립트를 분석하여 자기 복제 동작을 수행하는 코드 패턴들을 트리 형태로 정리하고, 이것을 정의된 문법에 따라 기술함으로써 완성된 룰을 얻을 수 있게 된다.

<48> 예컨대, 이미 알려진 비주얼 베이직 악성 스크립트들로부터 전자우편을 이용한 다양한 자기 복제 패턴들을 추출하여 하나의 트리로 정리하면 도 7 과 같은 형태를 보이게 된다. 도 7 에 제시된 트리의 단말 노드는 변형 없이 그대로 매칭 룰로 이용될 수 있으므로, 중간 노드들의 의미를 고려하여 정의된 문법에 따라 표현하면 도 8 과 같은 형태의 관계 룰을 얻을 수 있다.

<49> 도 7 에 나타난 룰 기술 중, 매칭 룰에서 사용된 '*" 는 어떠한 토큰에도 부합될 수 있는 와일드카드(wildcard)를 의미한다. 또한, 조건절에서 룰의 존재 여부만을 검사하는 관계 룰에서는 동작절의 우변에 별도의 룰 ID를 기술하지 않아도 조건을 만족하는 룰의 룰 변수로 인식하여 동작하게 된다. 예컨대, R4의 경우, M2의 룰이 만족됨으로 인해 조건식이 만족되었다면 동작절은 '$1 = M2.$1' 으로 해석되고, R6의 룰이 만족됨으로 인한 것이라면 '$1 = R6.$1' 으로 해석되므로 간략한 룰 기술이 가능하게 된다.

<50> 이와 같은 룰 기술을 통해 비주얼 베이직 스크립트 뿐 아니라, 다른 스크립트 언어로 작성된 행위 패턴도 동일한 방법으로 감지할 수 있다. 실제로 도 7 에 제시된 매칭 룰의 R3, R5, R8, R9의 첫 번째 토큰인 'Set' 만을 제거하면 자바스크립트에서 메일을 통한 자기복제 행위를 감지할 수 있다. 객체의 대입에는 반드시 'Set' 문장을 사용하여야 하는 비주얼 베이직 스크립트와는 달리, 자바 스크립트는 일반적인 형태의 대입문으로 이것이 가능하므로, 이러한 문법상의 차이만을 고려해 주면 관계 룰의 수정 없이도 동일한 감지 동작을 수행할 수 있다.

<51> 이번에는 스크립트 코드 삽입에 대해 상세히 설명하기로 한다. 대상 스크립트가 주어지면 악성 행위를 구성하는 메소드 시퀀스에 속한 메소드 호출 전후에, 파라미터와 리턴값을 버퍼에 저장하고 악성 행위를 검사하는 함수를 호출하는 코드가 삽입되어야 한다. 이는 스크립트 변환기에 의해 이루어지며, 코드 삽입이 이루어진 후의 메소드 호출 코드는 도 9 와 같이 변경된다. 도 9 에서 2행의 FSO.GetFile이 검사 대상인 메소드이고, 1행과 3행이 이 메소드의 동작을 검사하기 위해 삽입된 코드이다. RuleBase 객체는 룰 정의와 이에 의한 악성 행위 감지 루틴을 제공하는 객체이며, 그 구현은 코드 삽입 단계가 끝나면 수정된 코드의 뒷부분에 덧붙여진다. 이 객체의 코드는 자체 진단 루틴 생성기에 의해 생성되는데, 다음에 상세히 설명하기로 한다.

<52> 메소드 호출을 검사하기 위해 사용되는 메소드는 도 9 에서와 같이 SetVal과 Check 뿐이며, SetVal 은 배열로 구성된 버퍼의 주어진 위치에 해당값을 대입하고, Check는 버퍼의 내용을 참조하여 룰에 따라 검사를 수행하는 진단 엔진의 역할을 담당하게 된다. 이때, 사용하는 버퍼는 다수의 값들을 저장할 수 있도록 배열로 구성되어 있으므로 해당하는 매칭 룰의 이름 외에도, 모든 파라미터와 리턴값들이 하나의 배열에 저장된다. 비주얼 베이직이나 자바스크립트 등과 같은 많은 스크립트 언어에서는 일반적인 프로그래밍 언어에서의 어떠한 형(type)도 제약 없이 하나의 변수에 담길 수 있어서, 별도의 구조체를 이용하지 않고 배열만으로 동일할 작업을 수행할 수 있게된다. 버퍼를 구성하는 배열의 각 위치에 저장되는 자료의 의미는 다음의 표 2와 같다.

<53>

【표 2】

배열 위치	의 미	비 고
0	일치한 매칭 룰의 이름	문자열
1	리턴값	이름이 아닌 실행시의 값이 저장됨
2	호출된 메소드를 제공하는 객체	
3 이상	파라미터	

<54> 스크립트 변환기의 동작을 정리하면 다음과 같다. 첫번째로, 룰 기술 파일로부터 매칭 룰을 로드한다. 두번째로, 악성 행위 감지 루틴을 초기화하는 문장을 출력한다 (RuleBase 객체 초기화 메소드 호출). 세번째로, 주어진 스크립트의 모든 문장에 대해 다음의 작업을 수행한다. 즉, 한 문장을 읽어 읽은 문장이 매칭 룰과 일치하면 읽은 문장의 전후에 파라미터와 리턴값을 저장하고 자기진단 루틴을 호출하는 문장을 덧붙여 출력한다. 그러나, 읽은 문장이 매칭 룰과 일치하지 않으면 읽은 문장을 그대로 출력한다. 네번째로, 자체 진단 루틴 생성기로부터 얻어진 악성 행위 감지 루틴 코드(RuleBase 클래스 코드)를 추가한다.

<55> 이번에는 악성 행위 감지 루틴의 추가에 대해 상세히 설명하기로 한다. 메소드 호출 정보를 버퍼에 삽입하고 감지를 수행하는 악성 행위 감지 관련 루틴들은 하나의 클래스로 묶여질 수 있는데, 이는 상기에서 언급한 RuleBase 클래스이다. 이 클래스가 제공하는 공개(public) 메소드는 다음의 표 3 과 같다.

<56> 【표 3】

메 소 드	내 용
init	클래스 초기화
SetVal(pos, value)	value 값을 버퍼의 pos 번째에 대입
Check	악성 행위 여부 검사

<57> 코드 삽입 단계를 거치게 되면 매칭 룰에 기술된 형태를 가진 메소드 호출 문장 전후에 SetVal 메소드를 사용하여 검사에 필요한 값을 버퍼에 저장하고, Check 메소드를 호출하는 코드가 삽입된다. 실제로, Check 메소드는 단지 버퍼의 내용을 참조하여 어떠한 매칭 룰로 인해 자신이 호출되었는가를 알아낸 뒤, 해당 매칭 룰을 구현한 메소드를 호출하는 진입점(entry point)의 역할만을 수행한다. 즉, 각각의 룰은 악성 행위 탐지 클래스에 속한 하나의 내부 메소드(private method)로 나타나며, 자체 진단 루틴 생성기가 룰 정의를 참조하여 각각의 메소드를 자동적으로 생성하게 된다.

<58> 이때, 각각의 룰이 구현된 메소드의 내용은 매칭 룰과 관계 룰의 경우가 다르다. 매칭 룰은 이미 해당 문장이 주어진 형식과 일치할 경우에만 수행되므로, 아무런 조건 없이 해당 룰에 대한 매칭이 일어났음을 기록하기 위해 하나의 룰 인스턴스(instance)를 생성하고 상위 룰을 검사한다. 룰 인스턴스는 해당 룰에 관련된 정보를 담고 있는 데이터 구조이며 주어진 조건이 만족될 때 생성된다. 매칭 룰의 인스턴스에 저장되는 정보는 $1, $2와 같은 매칭 룰 변수의 값들이므로, 생성되는 인스턴스의 적절한 위치에 버퍼의 값들을 대입하는 것만으로 인스턴스의 생성이 가능하다.

<59> 관계 룰의 동작은 기본적으로 매칭 룰과 동일하나 항상 조건식을 먼저 검사하고 이것이 만족되었을 경우에만 해당 룰의 인스턴스를 생성한 뒤 상위 룰을 검사한다는 점이 다르다. 여기에서 조건식의 만족이란 실제로는 해당 조건식을 만족시켜주는 룰 인스턴스들이 존재함을 의미한다. 또한, 상위 룰이란 해당 룰을 자신의 조건식에 담고 있는 룰을 말하며, 도 7 과 같은 트리 형태로 룰을 나타낼 때 해당 룰의 부모 노드에 위치한 룰을 지칭하므로, 룰 정의의 분석을 통해 실행 전에 모두 결정이 가능하다.

<60> 자체 진단 루틴 생성기의 동작을 정리하면 다음과 같다. 첫번째로, 매칭 룰과 관계 룰을 로드한다. 두번째로, 로드된 룰들을 분석하여 각 룰들의 상위 룰을 기록한다. 즉, 모든 관계 룰에 대하여 하나의 룰 Rc를 선택하고, 선택된 룰의 조건식에 나타나는 룰들의 집합 S를 구하여 S에 속한 모든 룰들에 대해서 상위 룰을 Rc로 기록한다. 세번째로, 모든 매칭 룰에 대해, 이에 대응하는 메소드를 생성한다. 이때, 메소드의 내용은 다음과 같다. 즉, 버퍼의 내용을 참조하여 새로운 룰 인스턴스를 생성하고, 상위 룰에 대응하는 메소드를 호출한다. 네번째로, 모든 관계 룰에 대해, 이에 대응하는 메소드를 생성한다. 이때, 메소드 내용은 다음과 같다. 즉, 룰 바디 부분 부분을 파싱(parsing)하여 스크립트 문법에 맞게 변환하고, 상위 룰에 대응하는 메소드를 호출한다.

<61> 도 10 은 이렇게 생성된 악성 행위 감지 루틴에 의해 스크립트의 실행 중에 이루어지는 룰 인스턴스의 생성의 실예이다. 좌측에 제시한 스크립트의 각 행이 수행될 때마다 생성되는 룰 인스턴스를 우측에 도시하고 있으며, 각 룰 인스턴스의 이름 뒤에 붙여진 필드는 해당 룰 정의에서 $1, $2와 같이 기술된 룰 변수의 값을 의미한다.

【발명의 효과】

<62> 이상 설명한 바와 같이, 코드 삽입 기법을 이용한 악성 스크립트 감지 방법은 감지 루틴이 스크립트 실행 중에 동작함으로써 동적으로 결정되는 리턴값과 파라미터까지 검사할 수 있어서 감지 정확도를 향상시킬 수 있다. 또한, 외부로부터 유입된 스크립트에만 코드가 삽입됨으로써 불필요한 오버헤드를 발생시키지 않으며, 일단 변형된 코드는 별도의 안티바이러스가 설치되지 않은 시스템에서도 자체 감지를 수행함으로써 전파 억

제 효과를 얻을 수 있다. 한편, 특정 스크립트 언어에 국한되지 않고 사용하는 룰 집합을 다르게 정의함으로써 다수의 스크립트 언어에 동일하게 적용할 수 있다.

【특허청구범위】

【청구항 1】

악성 스크립트를 감지하는 방법에 있어서,

매칭 룰(Matching Rules)과 관계 룰(Relation Rules)을 포함한 룰(Rules) 기반의 메소드 호출 시퀀스 탐지를 채용하여 호출 시퀀스에 속한 각각의 문장에 관련된 값들을 검사하되,

원본 스크립트의 메소드 호출 문장 전후에 자체 진단 루틴(악성 행위 감지 루틴) 호출 문장을 삽입하는 단계; 및

상기 원본 스크립트에 삽입된 자체 진단 루틴을 통해 실행시 악성 코드를 감지하는 단계를 포함한 것을 특징으로 하는 코드 삽입 기법을 이용한 악성 스크립트 감지 방법.

【청구항 2】

제 1 항에 있어서,

상기 자체 진단 루틴 호출 문장은,

상기 메소드 호출 문장이 상기 매칭 룰에 기술된 내용과 일치시 메소드 호출 문장 전후에 삽입되는 파라미터와 리턴값 저장, 및 진단 엔진을 호출하는 문장들로 구성되며,

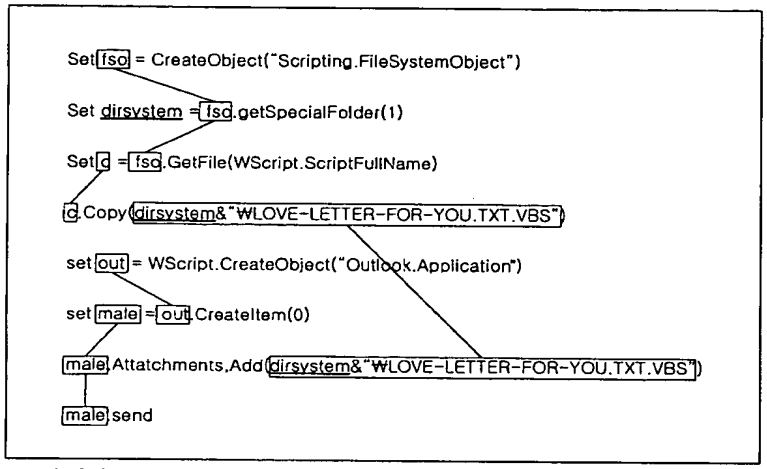
상기 자체 진단 루틴은,

상기 매칭 룰과 일치하는 형태의 메소드 호출시 수행되어 해당 매칭 룰에 관련된 관계 룰을 실행하여 메소드 호출 시퀀스의 악성 행위 구성 여부를 탐지하는

룰 기반의 진단 엔진과, 상기 진단 엔진이 사용할 수 있는 버퍼에 상기 매칭 룰을 만족하는 메소드 호출 문장의 파라미터와 리턴값을 저장하는 메소드들을 포함한 것을 특징으로 하는 코드 삽입 기법을 이용한 악성 스크립트 감지 방법.

【도면】

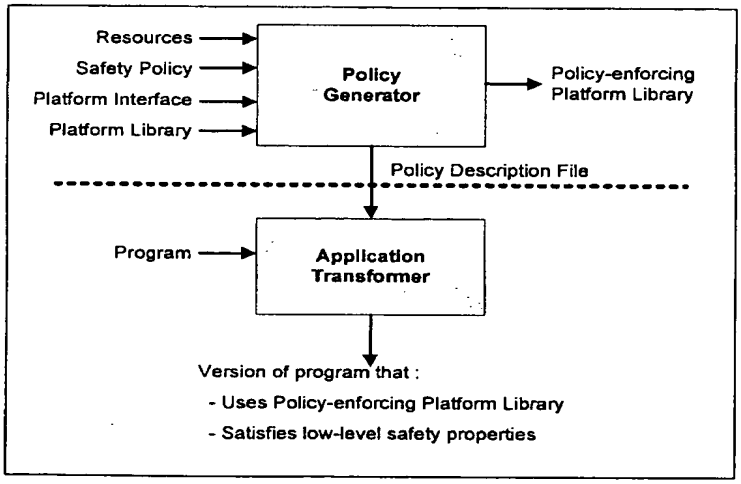
【도 1】



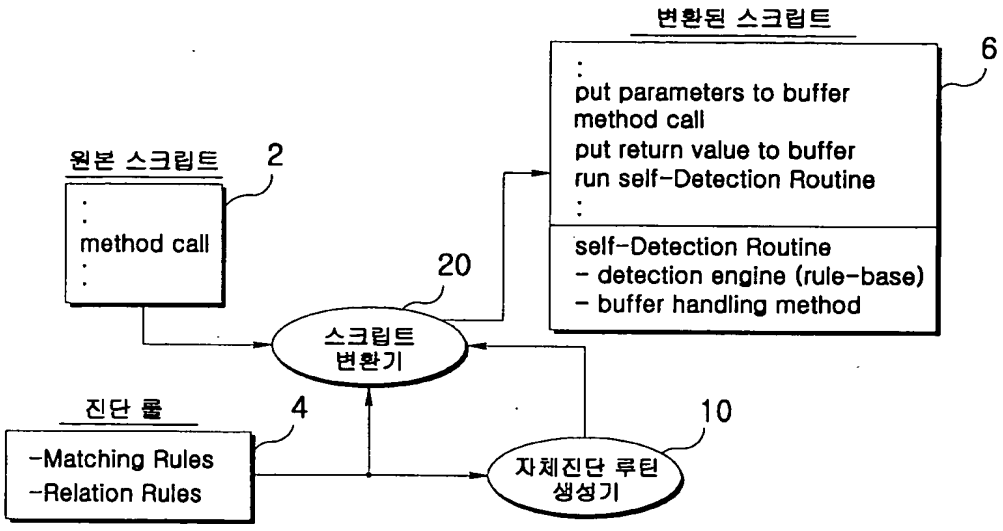
【도 2】

```
M1 : Set $1 = CreateObject(Scripting.FileSystemObject)
M2 : Set $1 = $2.GetFile(WScript.ScriptFullName)
M3 : $1.Copy($2)

M4 : set $1 = WScript.CreateObject(Outlook.Application)
M5 : set $1 = $2.CreateItem(0)
M6 : $1.Attatchments.Add($2)
M7 : $1.send

R1 : (M1.$1 == M2.$2) && (M2.$1 == M3.$1)
       => $1 = M3.$2                  // code writing
R2 : (M4.$1 == M5.$2) && (M5.$1 == M6.$1) && (M5.$1 == M7.$1)
       => $1 = M6.$2                  // send mail
R3 : R1.$1 == R2.$1
       => $1 = true                   // A Malicious Behavior is Detected!
```

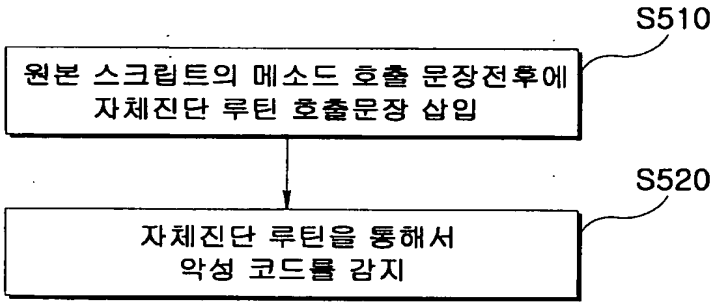
【도 3】



【도 4】



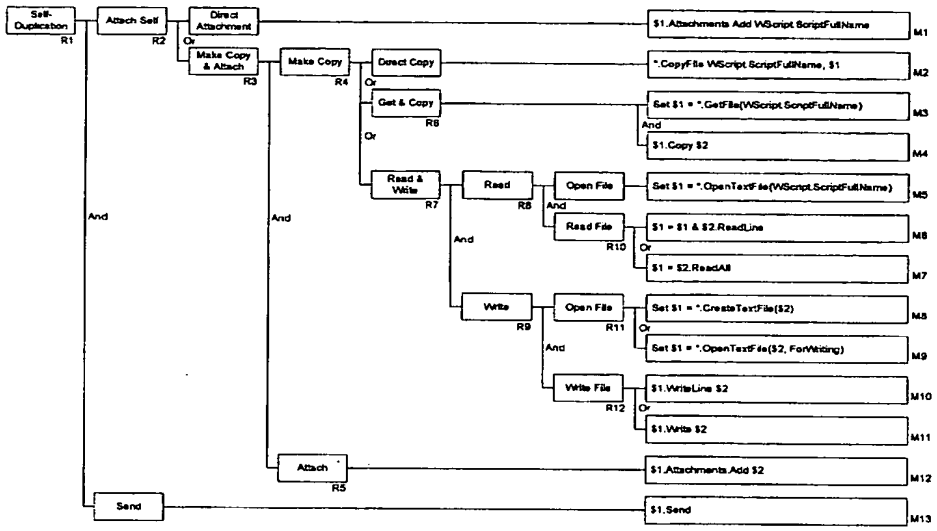
【도 5】



【도 6】

```
rule_description ::= { rule }1.
rule ::= rule_identifier : rule_body
rule_body ::= matching_rule_body | { relation_rule_body }1.
matching_rule_body ::= script statement with variable_string
variable_string ::= variable | *
variable ::= $ { digit }1.
relation_rule_body ::= condition_phrase => action_phrase
condition_phrase ::= condition_expr { logical_operator condition_expr }0.
condition_expr ::= rule_identifier | local_variable string_compare_operator local_variable
logical_operator ::= && | ||
string_compare_operator ::= < | ==
action_phrase ::= action_stmt { , action_stmt }0.
action_stmt ::= variable = rvalue
rvalue ::= local_variable | variable | true | false
local_variable ::= rule_identifier . variable
```

【도 7】



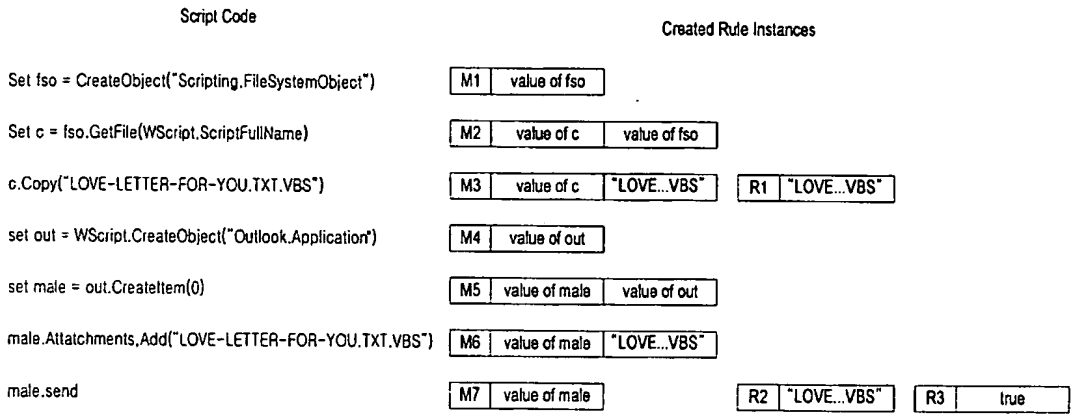
【도 8】

```
R1  : R2.$1 == M13.$1
      -> $1 = true
R2  : M1 || R3
      -> $1 = $1
R3  : R4.$1 == R5.$2
      -> $1 = R5.$1
R4  : M2 || R6 || R7
      -> $1 = $1
R5  : M12
      -> $1 = $1, $2 = $2
R6  : M3.$1 == M4.$1
      -> $1 = M4.$2
R7  : R8.$1 == R9.$2
      -> $1 = R9.$1
R8  : M5.$1 == R10.$2
      -> $1 = R10.$1
R9  : R11.$1 == R12.$1
      -> $1 = R11.$2, $2 = R12.$2
R10 : M6 || M7
      -> $1 = $1, $2 = $2
R11 : M8 || M9
      -> $1 = $1, $2 = $2
R12 : M10 || M11
      -> $1 = $1, $2 = $2
```

【도 9】

```
RuleBase.SetVal 0, "M2" : RuleBase.SetVal 2, FSO
Set c = FSO.GetFile(WScript.ScriptFullName)
RuleBase.SetVal 1, c : RuleBase.Check
```

【도 10】


Script Code
Created Rule Instances
Set fso = CreateObject("Scripting.FileSystemObject")
M1 value of fso
Set c = fso.GetFile(WScript.ScriptFullName)
M2 value of c value of fso
c.Copy("LOVE-LETTER-FOR-YOU.TXT.VBS")
M3 value of c "LOVE...VBS"
R1 "LOVE...VBS"
set out = WScript.CreateObject("Outlook.Application")
M4 value of out
set male = out.CreateItem(0)
M5 value of male value of out
male.Attatchments.Add("LOVE-LETTER-FOR-YOU.TXT.VBS")
M6 value of male "LOVE...VBS"
male.send
M7 value of male
R2 "LOVE...VBS"
R3 true